

## AI Lab 2

### 1) 8 puzzle problem using iteratively deepening depth first search

Code:

```
def print_puzzle(state):
    for i in range(3):
        print(state[i * 3:(i + 1) * 3])
    print()

def is_goal(state, goal):
    return state == goal

def find_blank_tile(state):
    return state.index(0)

def generate_moves(state):
    neighbors = []
    directions = {
        'up': -3,
        'down': 3,
        'left': -1,
        'right': 1
    }

    blank_index = find_blank_tile(state)

    for move, position_change in directions.items():
        new_blank_index = blank_index + position_change

        if move == 'up' and blank_index // 3 == 0:
            continue
        if move == 'down' and blank_index // 3 == 2:
            continue
        if move == 'left' and blank_index % 3 == 0:
            continue
        if move == 'right' and blank_index % 3 == 2:
            continue
```

```

        new_state = state[:]
        new_state[blank_index], new_state[new_blank_index] =
new_state[new_blank_index], new_state[blank_index]
        neighbors.append(new_state)

    return neighbors

```

```

def dfs(state, goal, depth_limit, visited_states, path):
    if state in visited_states:
        return False
    if is_goal(state, goal):
        return path + [state] # Return the path leading to the goal

    if depth_limit <= 0:
        return False

```

```

    visited_states.append(tuple(state))

```

```

    for neighbor in generate_moves(state):
        result = dfs(neighbor, goal, depth_limit - 1, visited_states, path + [state])
        if result: # If a solution is found
            return result

```

```

    visited_states.remove(tuple(state))
    return False # No solution found

```

```

def iddfs(start, goal, max_depth):
    for depth in range(max_depth + 1):
        visited_states = list() # Use a set for faster membership checks
        print(f"Trying depth limit: {depth}")
        path = dfs(start, goal, depth, visited_states, []) # Start with an empty path
        if path:
            print(f"Solution found at depth {depth} with path:")
            for state in path:
                print_puzzle(state) # Print each state in the solution path
            return True
    print("No solution found within the given depth limit.")

```

```
return False

start_state = [1, 2, 3, 5, 6, 0, 4, 7, 8]
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

max_depth = 20
iddfs(start_state, goal_state, max_depth)
```

Output:

```
  Solution found at depth 5 with path:
  [1, 2, 3]
  [5, 6, 0]
  [4, 7, 8]

  [1, 2, 3]
  [5, 0, 6]
  [4, 7, 8]

  [1, 2, 3]
  [0, 5, 6]
  [4, 7, 8]

  [1, 2, 3]
  [4, 5, 6]
  [0, 7, 8]

  [1, 2, 3]
  [4, 5, 6]
  [7, 0, 8]

  [1, 2, 3]
  [4, 5, 6]
  [7, 8, 0]

: True
```